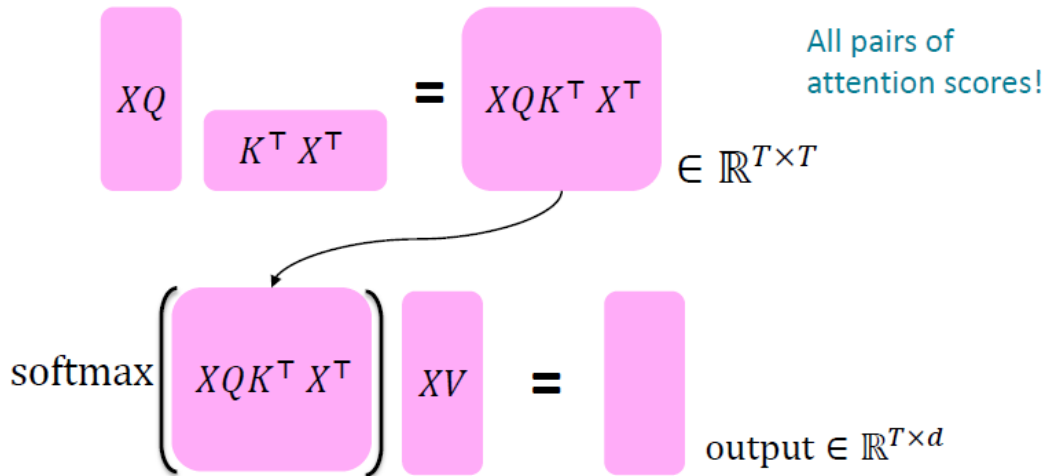


Transformers

W

Key-query-value attention

- Obtain q_t, v_t, k_t from X_t
- $q_t = W^q X_t; v_t = W^v X_t; k_t = W^k X_t$ (position encoding omitted)
 - W^q, W^v, W^k are learnable weight matrices
- $\alpha_{i,j} = \text{softmax}(q_i^\top k_j); \text{out}_i = \sum_k \alpha_{i,j} v_j$
- Intuition: key, query, and value can focus on different parts of input

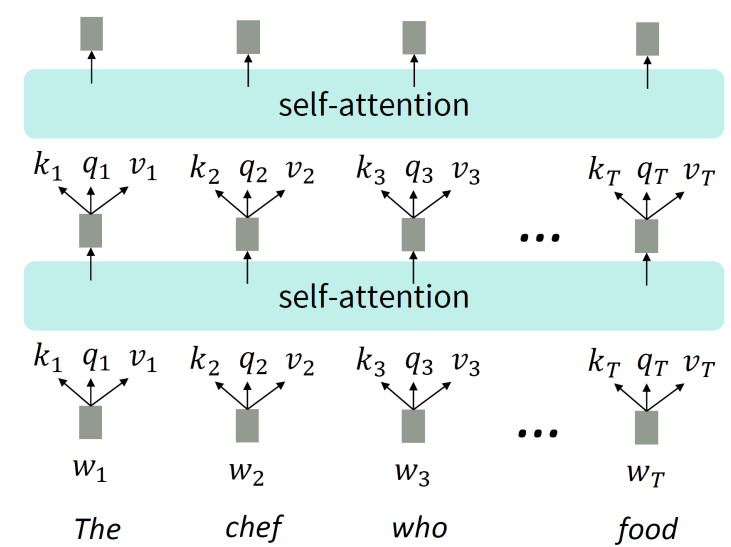
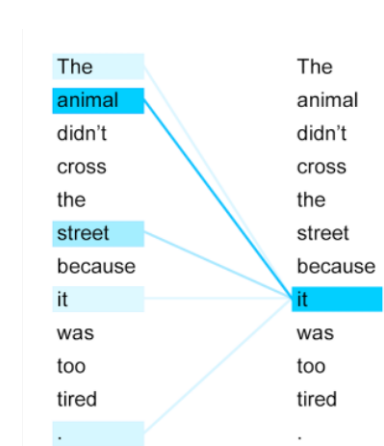


Attention is all you need (Vaswani '17)

- A pure attention-based architecture for sequence modeling
 - No RNN at all!
- Basic component: self-attention, $Y = f_{SA}(X; \theta)$
 - X_t uses attention on entire X sequence
 - Y_t computed from X_t and the attention output
- Computing Y_t

- Key $k_{t'}$, value $v_{t'}$, query q_t from X_t
 - $(k_t, v_t, q_t) = g_1(X_t; \theta)$
- Attention distribution $\alpha_{t,j} = \text{softmax}(q_t^\top k_j)$
 - Attention output $out_t = \sum_j \alpha_{t,j} v_j$
 - $Y_t = g_2(out_t; \theta)$

$$\text{Output}_{w_1} = \sum_{j=1}^T \text{softmax}\left(q_t^\top k_j\right) \cdot v_j$$



Issues of Vanilla Self-Attention

- Attention is order-invariant
- Lack of non-linearities
 - All the weights are simple weighted average
- Capability of autoregressive modeling
 - In generation tasks, the model cannot “look at the future”
 - e.g. Text generation:
 - Y_t can only depend on $X_{i < t}$
 - But vanilla self-attention requires the entire sequence

Position Encoding

- Vanilla self-attention

- $(k_t, v_t, q_t) = g_1(X_t; \theta)$

- $\alpha_{t,j} = \text{softmax}(q_t^\top k_j)$

- Attention output $out_t = \sum_j \alpha_{t,j} v_j$

No built-in understanding of sequence order

If you scrambled the words in the input, output would be the same, just also scrambled

Position Encoding

- Vanilla self-attention

- $(k_t, v_t, q_t) = g_1(X_t; \theta)$

- $\alpha_{t,j} = \text{softmax}(q_t^\top k_j)$

- Attention output $out_t = \sum_j \alpha_{t,j} v_j$

No built-in understanding of sequence order

If you scrambled the words in the input, output would be the same, just also scrambled

- Idea: position encoding:

- p_i : an embedding vector (feature) of position i

- $(k_t, v_t, q_t) = g_1([X_t, p_t]; \theta)$

- In practice: Additive is sufficient: $k_t \leftarrow \tilde{k}_t + p_t, q_t \leftarrow \tilde{q}_t + p_t, v_t \leftarrow \tilde{v}_t + p_t$;

$$(\tilde{k}_t, \tilde{v}_t, \tilde{q}_t) = g_1(X_t; \theta)$$

- p_t is only included in the first layer

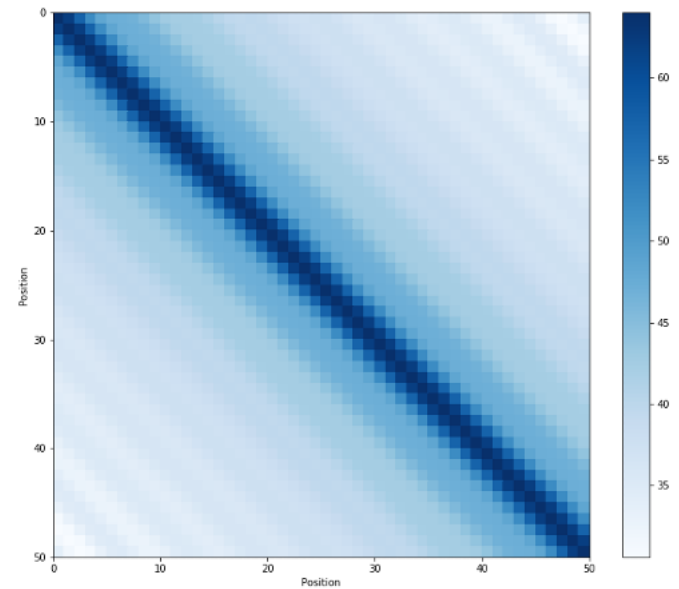
Position Encoding

OG approach of Vaswani et al. 2017

p_t design 1: Sinusoidal position representation

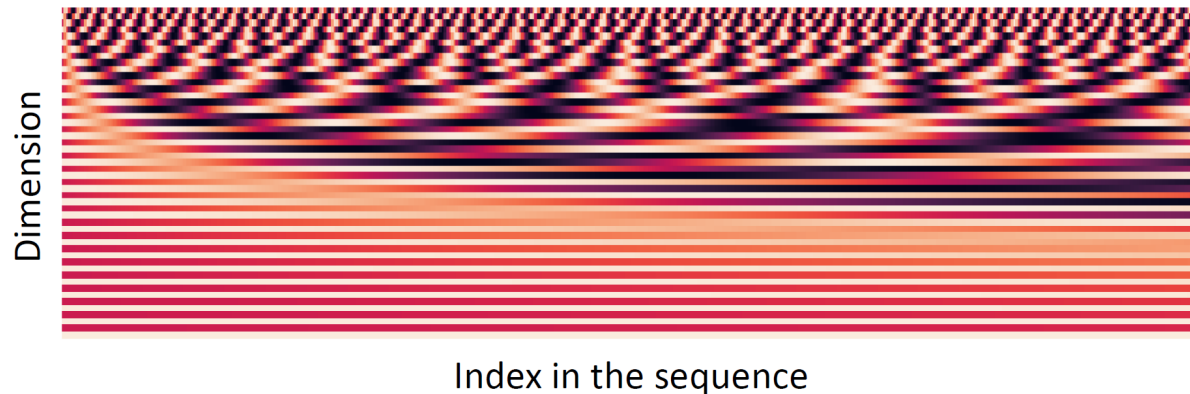
- Pros:
 - simple
 - naturally models “relative position”
 - Easily applied to long sequences
- Cons:
 - Not learnable
 - Generalization poorly to sequences longer than training data

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*d/2/d}) \\ \cos(i/10000^{2*d/2/d}) \end{pmatrix}$$



Heatmap of $p_i^T p_j$

Dot product strongest for closer words, falls off with predictable pattern



Each column is the encoding of word 0, 1, 2, ...

Position Encoding

p_t design 2: Learned representation

- Assume maximum length L , learn a matrix $p \in \mathbb{R}^{d \times T}$, p_t is a column of p
- Pros:
 - Flexible
 - Learnable and more powerful
- Cons:
 - Need to assume a fixed maximum length L
 - Does not work at all for length above L

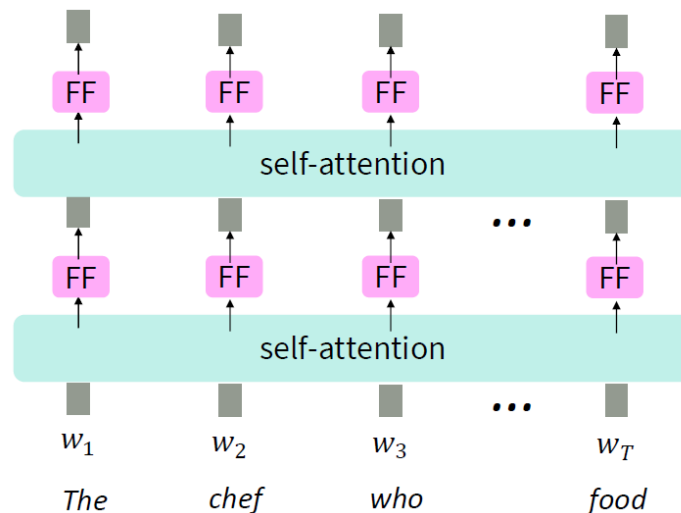
Combine Self-Attention with Nonlinearity

- Vanilla self-attention
 - No element-wise activation (e.g., ReLU, tanh)
 - Only weighted average and softmax operator
- Fix:
 - Add an MLP to process out_i
 - $m_i = MLP(out_i) = W_2 \text{ReLU}(W_1 out_i + b_1) + b_2$
 - Usually do not put activation layer before softmax

Why?

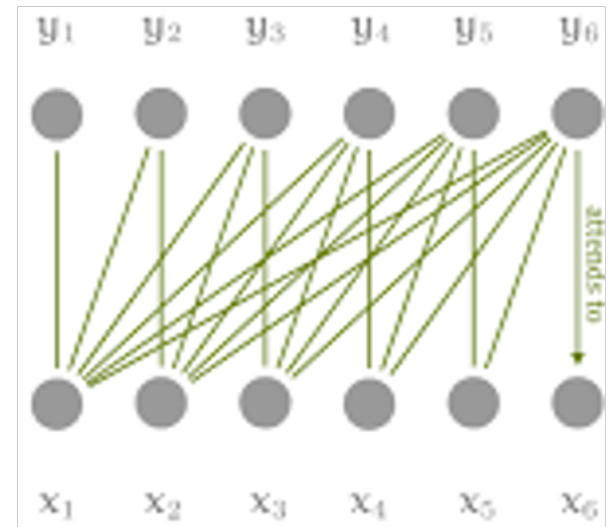
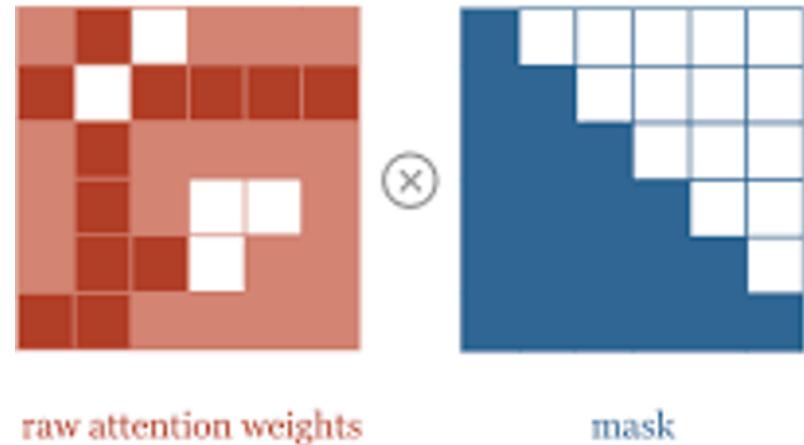
Even though softmax is linear, without this path from $X \rightarrow out$ is linear.

Wouldn't get you universal approximation



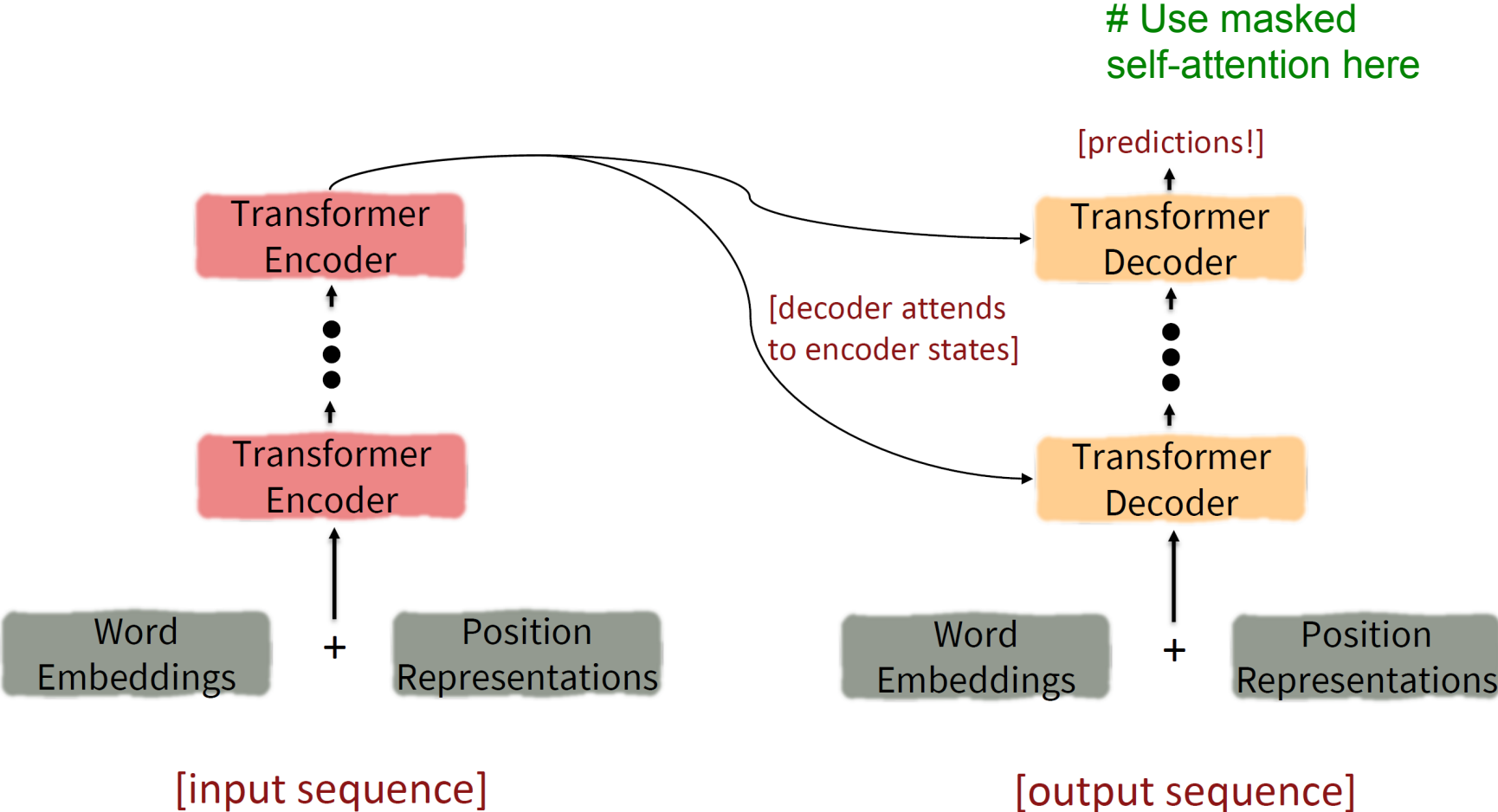
Masked Attention

- In language model decoder: $P(Y_t | X_{i < t})$
 - out_t cannot look at future $X_{i > t}$
- Masked attention
 - Compute $e_{i,j} = q_i^\top k_j$ as usual
 - Mask out $e_{i>j}$ by setting $e_{i>j} = -\infty$
 - $e \odot (1 - M) \leftarrow -\infty$
 - M is a fixed 0/1 mask matrix
 - Then compute $\alpha_i = \text{softmax}(e_i)$
 - Remarks:
 - $M = 1$ for full self-attention
 - Set M for arbitrary dependency ordering



Transformer

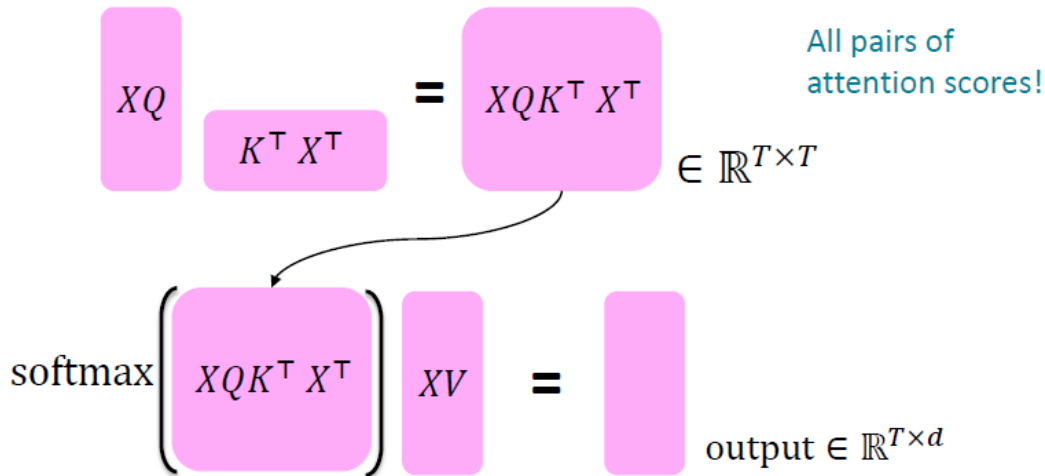
Transformer-based sequence-to-sequence modeling



Queries come from encoder, Keys and Values from previous decoder layer

Key-query-value attention

- Obtain q_t, v_t, k_t from X_t
- $q_t = W^q X_t; v_t = W^v X_t; k_t = W^k X_t$ (position encoding omitted)
 - W^q, W^v, W^k are learnable weight matrices
- $\alpha_{i,j} = \text{softmax}(q_i^\top k_j); \text{out}_i = \sum_k \alpha_{i,j} v_j$
- Intuition: key, query, and value can focus on different parts of input



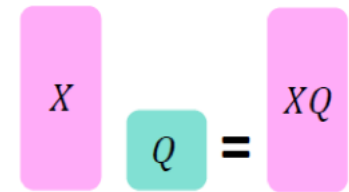
Multi-headed attention

- Standard attention: single-headed attention
 - $X_t \in \mathbb{R}^d, Q, K, V \in \mathbb{R}^{d \times d}$
 - We only look at a single position j with high $\alpha_{i,j}$
 - What if we want to look at different j for different reasons?
- Idea: define h separate attention heads
 - h different attention distributions, keys, values, and queries
 - $Q^\ell, K^\ell, V^\ell \in \mathbb{R}^{d \times \frac{d}{h}}$ for $1 \leq \ell \leq h$
 - $\alpha_{i,j}^\ell = \text{softmax}((q_i^\ell)^\top k_j^\ell); \text{out}_i^\ell = \sum_j \alpha_{i,j}^\ell v_j^\ell$

#Params Unchanged!

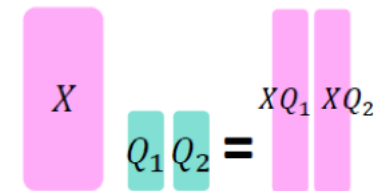
Single-head attention

(just the query matrix)



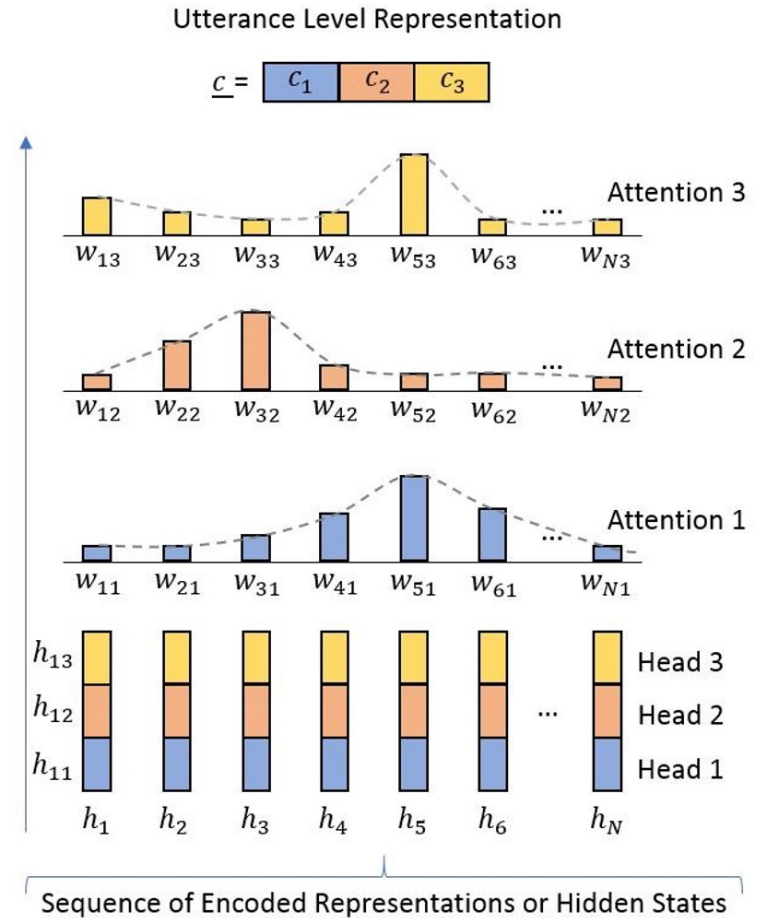
Multi-head attention

(just two heads here)



Multi-headed attention

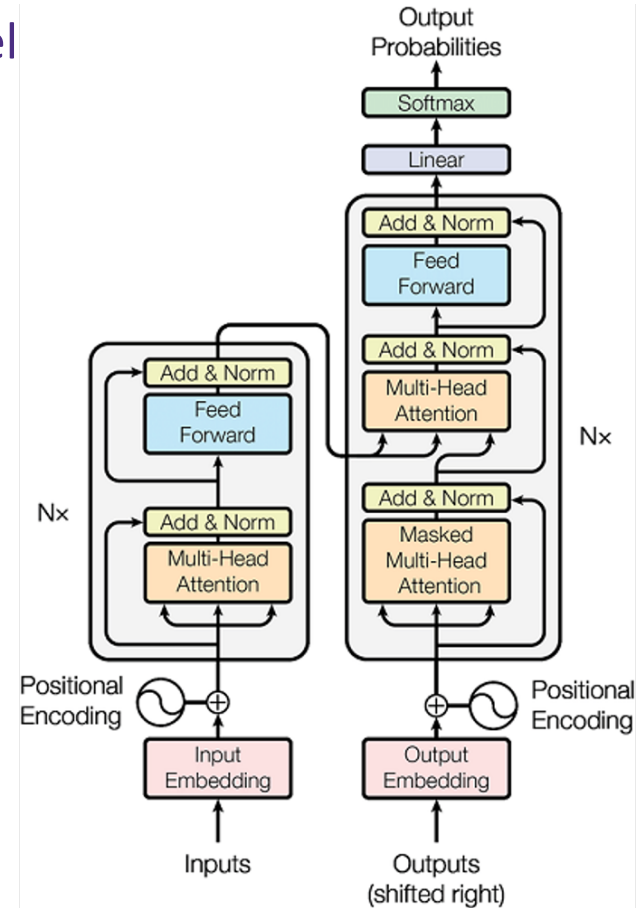
- Standard attention: single-headed attention
 - $X_t \in \mathbb{R}^d, Q, K, V \in \mathbb{R}^{d \times d}$
 - We only look at a single position j with high $\alpha_{i,j}$
 - What if we want to look at different j for different reasons?
- Idea: define h separate attention heads
 - h different attention distributions, keys, values, and queries
 - $Q^\ell, K^\ell, V^\ell \in \mathbb{R}^{d \times \frac{d}{h}}$ for $1 \leq \ell \leq h$
 - $\alpha_{i,j}^\ell = \text{softmax}((q_i^\ell)^\top k_j^\ell); out_i^\ell = \sum_j \alpha_{i,j}^\ell v_j^\ell$



Transformer

Transformer-based sequence-to-sequence model

- Basic building blocks: self-attention
 - Position encoding
 - Post-processing MLP
 - Attention mask
- Enhancements:
 - Key-query-value attention
 - Multi-headed attention
 - Architecture modifications:
 - Residual connection
 - Layer normalization



Transformer

Machine translation with transformer

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8	$2.3 \cdot 10^{19}$	

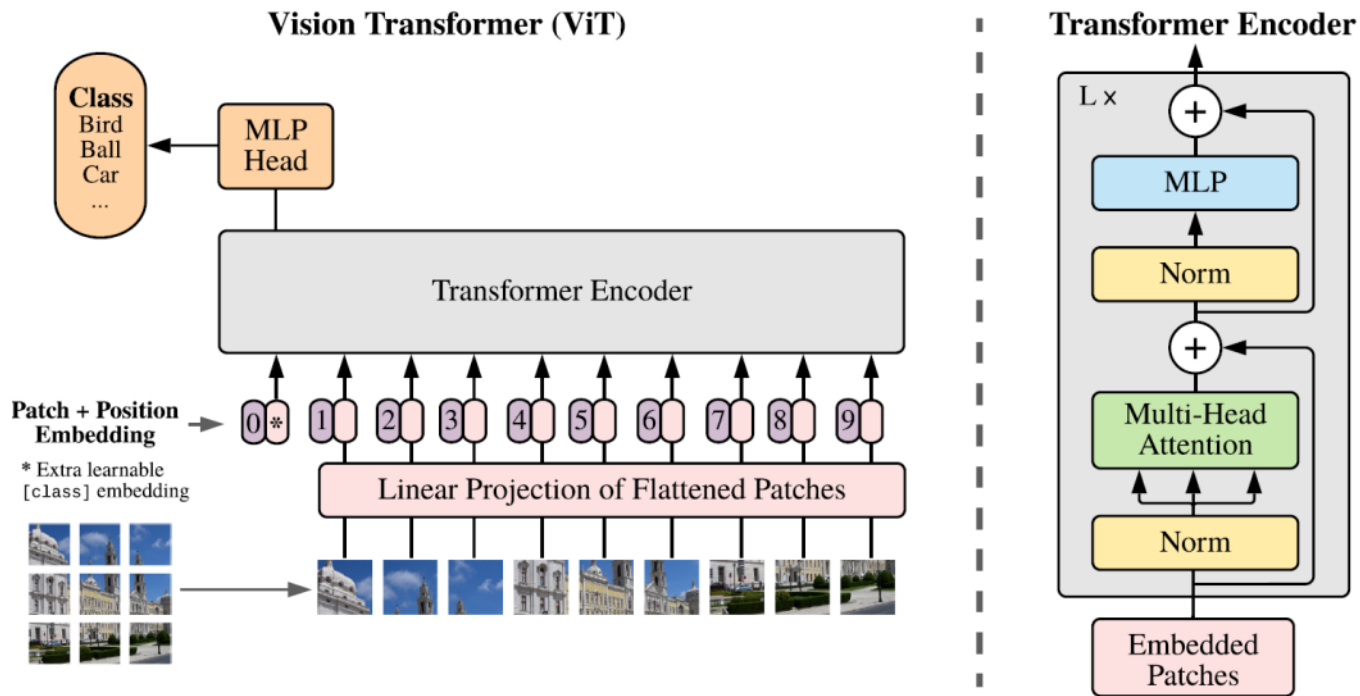
Transformer

- Limitations of transformer: Quadratic computation cost
 - Linear for RNNs
 - Large cost for large sequence length, e.g., $L > 10^4$

- Follow-ups:
 - Large-scale training: transformer-XL; XL-net ('20)
 - Projection tricks to $O(L)$: Linformer ('20)
 - Math tricks to $O(L)$: Performer ('20)
 - Sparse interactions: Big Bird ('20)
 - Deeper transformers: DeepNet ('22)

Transformer for Images

- Vision Transformer ('21) # Cited 96,066 times
 - Decompose an image to 16x16 patches and then apply transformer encoder



- **Big lesson:** with enough data, you don't need the hand-designed structure of CNNs, and that can actually hold you back. "Scale is all you need"

Transformer for Images

- Swin Transformer ('21) # SWin = “Shifted Window”
 - Build hierarchical feature maps at different resolution
 - Self-attention only within each block
 - Shifted block partitions to encode information between blocks

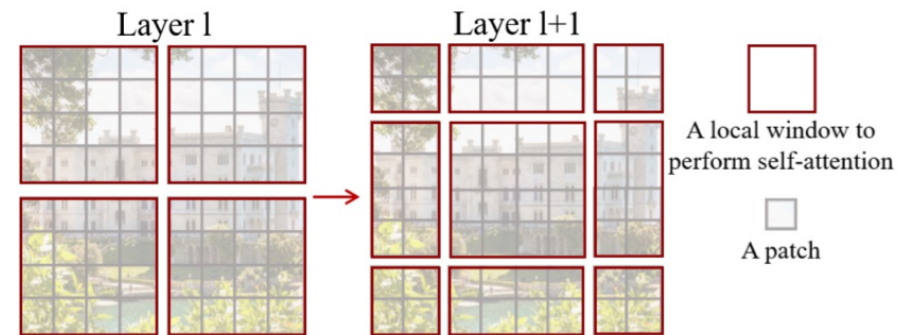
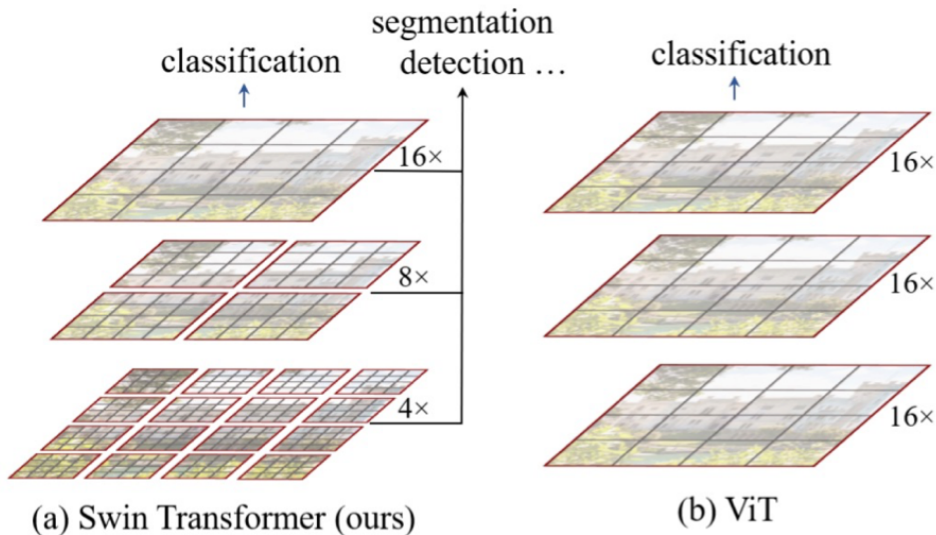


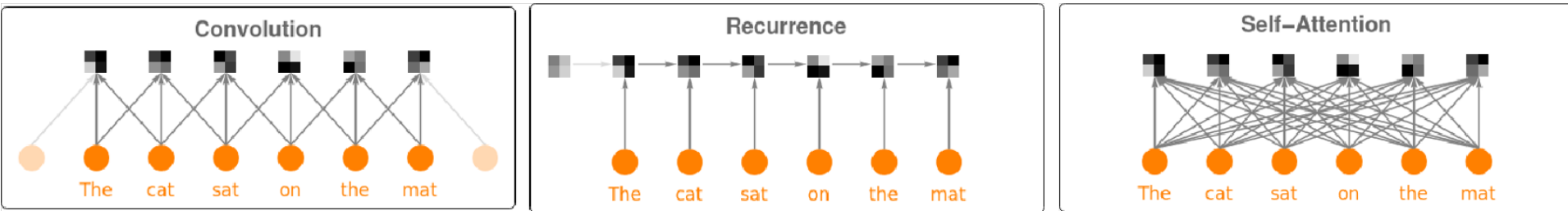
Figure 2. An illustration of the *shifted window* approach for com-

Why was this needed?

Attention is quadratic cost over patches. With N patches, cost is $O(N^2)$

Gives back the ability to learn features at different scales

CNN vs. RNN vs. Attention

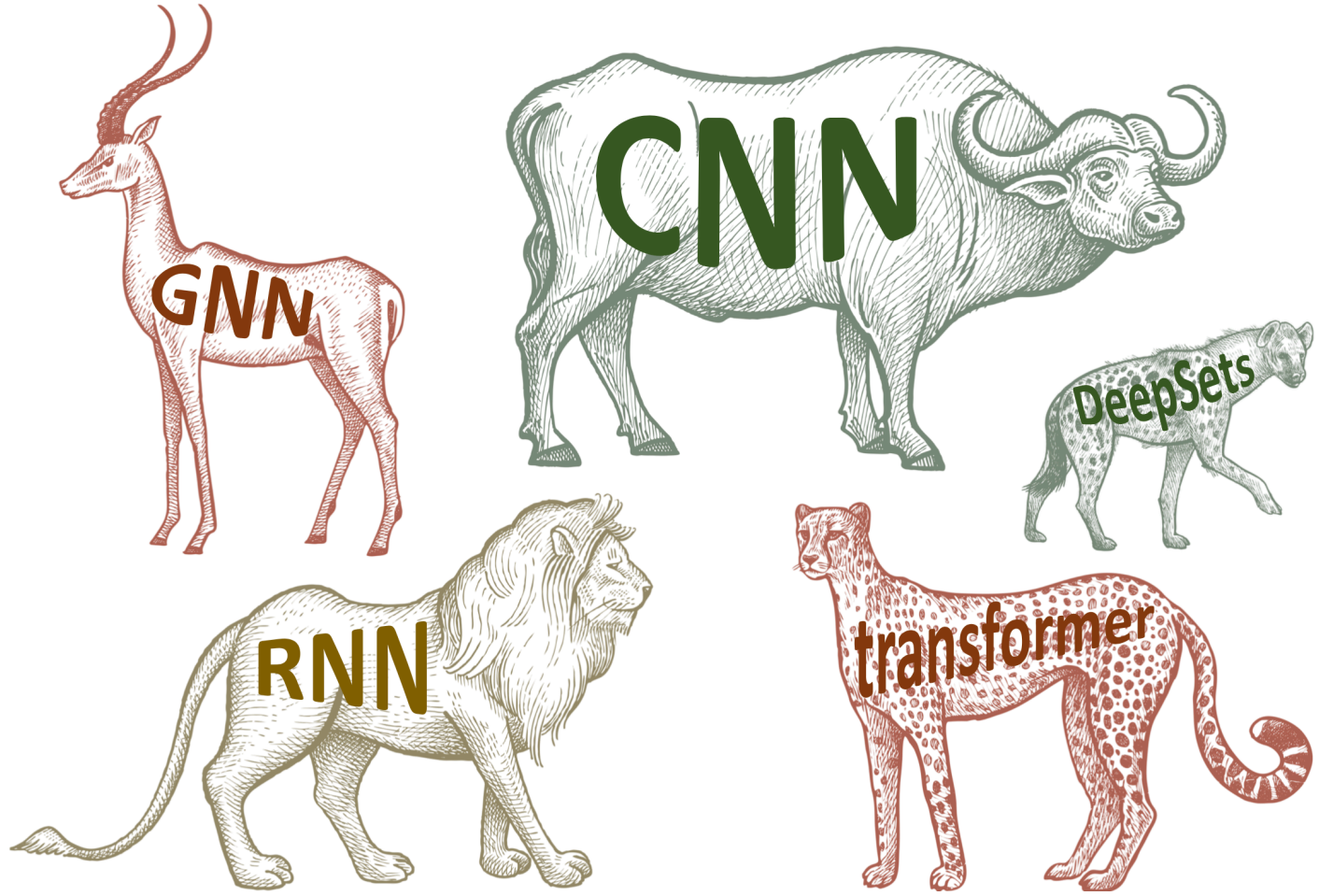


- **CNNs:** built in equivariances; re-use filters through image. Assume local structure matters most, deeper layers handle larger structure. Computationally efficient, easier to learn.
- **RNNs:** when processing sequences occurring over time, retain information in memory, understand ordering. Sequential information bottleneck (encoding the whole sequence into one vector) means poor performance on long sequences.
- **Attention:** any two positions could be relevant to each other; let the model learn which. Maximally flexible, fully parallelizable... but $O(N^2)$ computation cost, no built-in notion of ordering, no built-in priors so bad for small data.

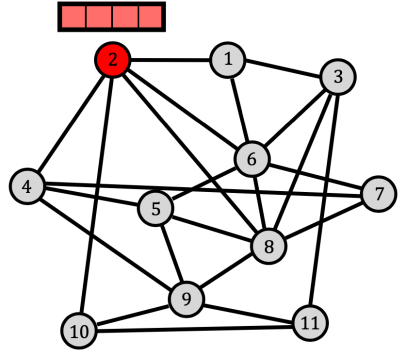
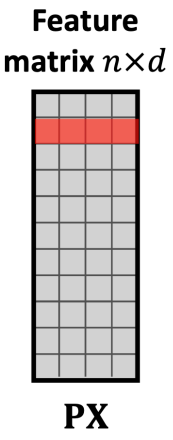
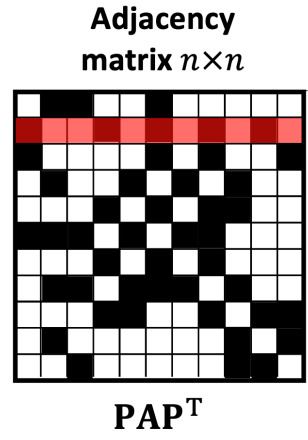
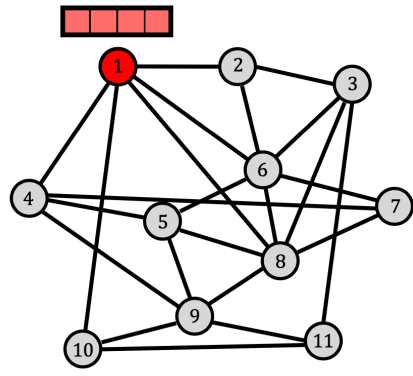
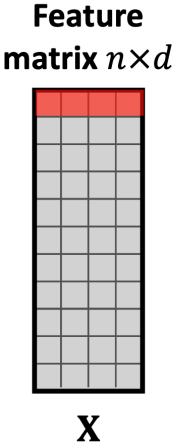
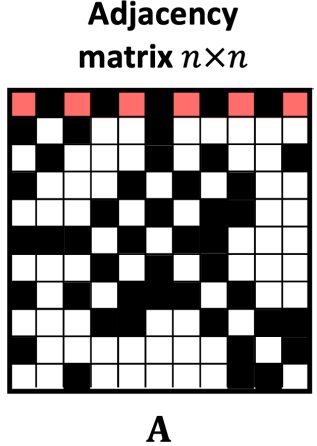
Summary

- Language model & sequence to sequence model:
 - Fundamental ideas and methods for sequence modeling
- Attention mechanism
 - So far the most successful idea for sequence data in deep learning
 - A scale/order-invariant representation
 - Transformer: a fully attention-based architecture for sequence data
 - Transformer + Pretraining: the core idea in today's NLP tasks
- LSTM is still useful in lightweight scenarios

Other architectures



Graph Neural Networks

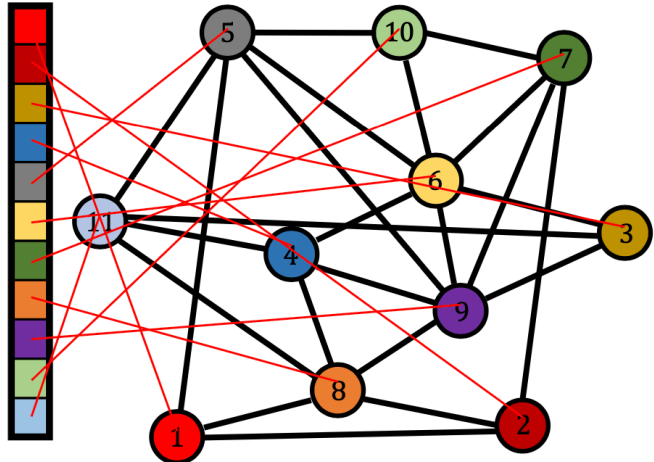


arbitrary ordering of nodes

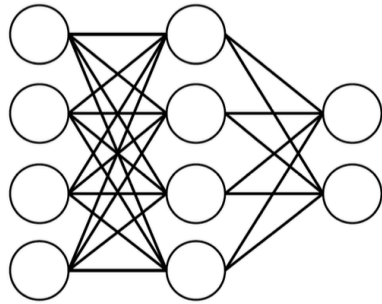
Graph Neural Networks

permutation-equivariant

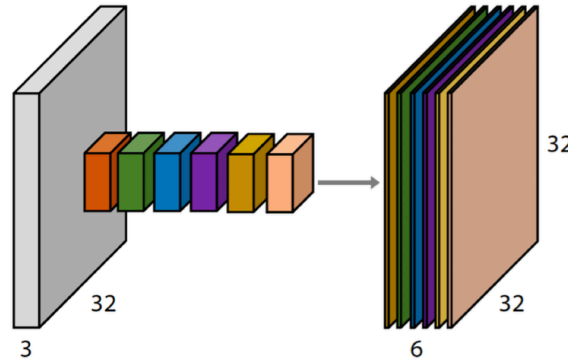
$$\mathbf{F}(\mathbf{P}\mathbf{X}, \mathbf{P}\mathbf{A}\mathbf{P}^T) = \mathbf{P}\mathbf{F}(\mathbf{X}, \mathbf{A})$$



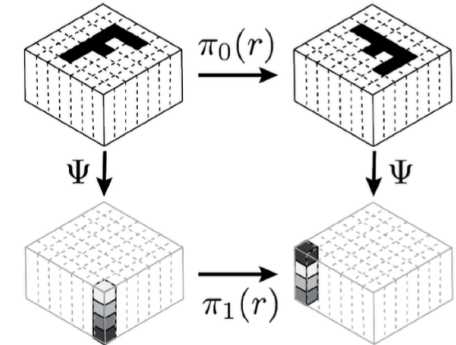
Geometric Deep Learning



Perceptrons
Function regularity



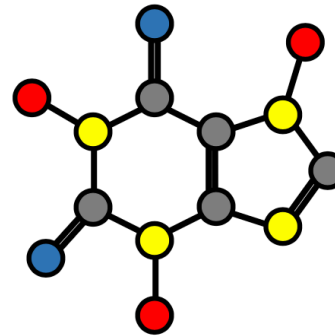
CNNs
Translation



Group-CNNs
Translation+Rotation



DeepSets / Transformers
Permutation



GNNs
Permutation



Intrinsic CNNs
Local frame choice